Deploying OSK on Low-resource Mobile Devices

Gildas Avoine[1], Muhammed Ali Bingöl[2,3], Xavier Carpent[1], Süleyman Kardaş[2,3]

[1] Université catholique de Louvain ICTEAM Institute B-1348, Louvain la Neuve, Belgium
[2] TÜBİTAK BİLGEM UEKAE Gebze, Kocaeli, Turkey
[3] Sabancı University, Faculty of Engineering and Natural Sciences, İstanbul, TR-34956, Turkey

Abstract. It is a popular challenge to design authentication protocols that are both privacy-friendly and scalable. A large body of literature in RFID is dedicated to that goal, and many inventive mechanisms have been suggested to achieve it. However, to the best of our knowledge, none of these protocols have been tested so far in practical scenarios. In this paper, we present an implementation of the OSK protocol, a scalable and privacy-friendly authentication protocol, using a variant by Avoine and Oechslin that accommodates it to time-memory trade-offs. We show that the OSK protocol is suited to certain real-life scenarios, in particular when the authentication is performed by low-resource mobile devices. The implementation, done on an NFC-compliant cellphone and a ZC7.5 contactless tag, demonstrates the practicability and efficiency of the OSK protocol and illustrates that privacy-by-design is achievable in constrained environments.

Keywords: RFID authentication, implementation, time-memory trade offs, privacy

1 Introduction

A major research topic in RFID is the development of authentication protocols that respect the privacy of the users, while still being efficient enough to be applicable in large-scale systems. When the time needed by the authentication process is not negligible, the user must hold the card steady in front of the reader until reception of an audio or visual signal. Long authentication processes are not practical in access control systems where delaying the customer flow is not acceptable for example in mass transportation or cultural events. It is generally agreed upon that approximately 200 milliseconds can be dedicated to grant or deny the access to a customer in a flow [10].

Classical challenge-response protocols such as ISO/IEC 9798-2.2 [14] can be privacy-friendly if the tag (prover) does not send its identifier in the clear to the reader (verifier). In such a case, the reader must find the tag identifier by performing an exhaustive search in its database. For example, in a system managing 2^{20} tags and a reader capable of performing 2^{20} cryptographic operations per second, the authentication of a tag takes half a second on average, which is beyond the time threshold that can be allocated to the security operations.

Several protocols have been designed to provide privacy in the authentication. An additional property, named forward privacy, ensures that if a tag is compromised at one point, an adversary is not able to trace it in the past (given past communication traces). An example of such a protocol is the OSK protocol, proposed by Ohkubo, Suzuki and Kinoshita in [20]. It may be regarded as one of the most privacy-friendly protocol among the ones that allow for an efficient authentication procedure based on symmetric-key cryptography [1].

An implementation of this protocol was previously done in [6]. It uses rainbow tables accomodated for OSK, as proposed by Avoine and Oechslin in [5]. However, we show in this paper that in the setting of [6], a faster and simpler approach is viable, namely the full storage. Instead, we focus on systems with low-resource mobile readers, such as PDA's or NFC-compliant cellphones, and adapt this implementation to that context. We show that such a protocol with very good privacy properties is efficient enough to be used in practice, even in such constrained environments.

The structure of the paper is as follows. We present the OSK protocol in Sect. 2 and the adapted time-memory trade-off in Sect. 3. The method of Avoine and Oechslin for accommodating OSK for time-memory trade-offs is described in Sect. 4. We describe our implementation and discuss our results in Sect. 5. We finally conclude in Sect. 6.

2 Ohkubo, Suzuki, and Kinoshita's Protocol

2.1 Description

The OSK protocol is proposed by Ohkubo, Suzuki, and Kinoshita in [20]. It is one of the most well-known RFID-devoted authentication protocols and is the earliest one that achieves *forward privacy*[4]. In the RFID context, forward privacy is the property that guarantees the security of past

[4] It is also known as *backward untraceability* and used interchangeably in some papers [16, 18, 21].

interactions of a tag even if it is compromised at a later stage. Namely, the secret information of a tag $\mathcal{T}_i$ $(1 \leq i \leq n)$ is corrupted by an adversary at time t, the adversary can not associate any transaction with $\mathcal{T}_i$ at any time $t' < t$.

In the OSK protocol, each tag $\mathcal{T}_i$ has an initial secret S_i^0 that is updated after each authentication query. The update consists in hashing the current secret with the one-way function $\mathcal{H}$. Upon reception of the authentication query, the tag answers by hashing the current secret with a different[5] hash function, $\mathcal{G}$. Fig. 1 shows the OSK protocol.

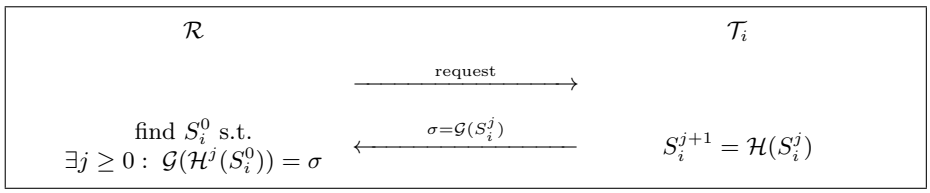


Fig. 1. OSK Protocol

System Setup. Each tag $\mathcal{T}_i$ of the system is initialized with a randomly chosen secret S_i^0. The n initial secrets are stored in a database, sometimes called back-end system. In some settings the back-end system and the reader are two different devices, connected in a way that is considered secure. In some other settings the back-end system is embedded in the readers.

Interrogation. When the tag is queried by a reader it answers with a response using the current secret such that $\sigma = \mathcal{G}(S_i^j)$ and also updates the secret immediately using a different hash function: $S_i^{j+1} = \mathcal{H}(S_i^j)$.

Search & Identification. When receiving an answer the database searches for an initial secret S_i^0 that leads to σ. In other words, it checks whether there exists i and j such that $\mathcal{G}(\mathcal{H}^j(S_i^0)) = \sigma$. To do that, from each of the n initial secrets S_i^0, the reader computes the hash chains as shown in Fig. 2 until it finds a value matching σ, or until it reaches a given maximum limit L (the "lifetime" of a tag) on the chain length.

[5] Note that although these two functions need to be different, only one algorithm may be implemented on the tag, and an additional 1-bit input parameter used to select the function.

$$
\begin{array}{ccccccccc}
S_1^0 & \longrightarrow & r_1^0 & r_1^1 & r_1^2 & \dots & r_1^{L-1} & r_1^L \\
S_2^0 & \longrightarrow & r_2^0 & r_2^1 & r_2^2 & \dots & r_2^{L-1} & r_2^L \\
\dots & \longrightarrow & \dots & \dots & \dots & \dots & \dots & \dots \\
S_i^0 & \longrightarrow & \dots & \dots & \dots & \boxed{r_i^j = \mathcal{G}(\mathcal{H}^j(S_i^0))} & \dots & r_i^L \\
\dots & \longrightarrow & \dots & \dots & \dots & \dots & \dots & \dots \\
S_n^0 & \longrightarrow & r_n^0 & r_n^1 & r_n^2 & \dots & r_n^{L-1} & r_n^L
\end{array}
$$

Fig. 2. OSK table: chains of hashes in the OSK protocol.

The value $\sigma = \mathcal{G}(S_i^j)$ does not leak any information to an attacker on the secret of $\mathcal{T}_i$ when $\mathcal{G}$ and $\mathcal{H}$ behave as pseudo-random functions. However, given that the authentication process is bounded by L, the OSK protocol is prone to desynchronization when an adversary queries the tag more than L times. In such a case, the tag can no longer be authenticated and a privacy issue arises for a certain type of adversary, capable of detecting the success status of an authentication (see [15] for a discussion). Fortunately, the synchronization can be retrieved by the back-end without exchanging the tag; for example, the holder of a *desynchronized* tag can ask the system operator to recompute the chain of his tag.

Beside this desynchronization issue – and although the protocol is very efficient when all the tags are synchronized [20] – the worst-case complexity of the search procedure makes the protocol unsuitable for most practical applications.

2.2 Real-life Applications

We now discuss the possibility of implementing OSK in real-life applications. Throughout this article, we chose a system of $n = 2^{20}$ tags with a lifetime of $L = 2^7$, which are reasonable parameters and in accordance with [3].

Online Search. A naïve approach for the server is to only keep the initial secrets and recompute the $n \times L$ possibilities each time it receives a given σ. With a server capable of 2^{20} cryptographic hash operations per second, this takes 2^6 seconds ≈ 1 minute on average for these parameters. This is far beyond our limit of 200 milliseconds for a reasonable identification time.

Full Storage. The other extreme solution consists in storing all the chains in a table and letting the server perform a simple look-up whenever it receives σ. This solution has the advantage of requiring no cryptographic operation during the authentication, which makes the authentication very fast. Unfortunately, this approach has two major drawbacks.

First of all, a large memory is needed to store the table: given our parameters ($n = 2^{20}$ tags with a lifetime of $L = 2^7$, and a hash size of 128 bits), the full storage approach requires 2^{34} bits = 2 GB.[6] In a system where readers are permanently connected to the back-end server, requiring such a memory (RAM) for the server is not a major problem. However, in systems consisting of mobile readers sporadically connected to the database, the authentication material should be replicated in each of these low-resource devices. In such a scenario, this amount of memory is very large for small devices such as PDA's or handheld RFID readers, which typically have a memory of 128 MB. It might, however, be reasonable for more elaborate devices such as NFC-enabled smartphones, which have several gigabytes of flash memory.

A second issue is that, every now and then, the table needs to be either computed in a central server and uploaded on the smartphones, or computed by the smartphones themselves after reception of the first column of the table. This might take a significant time for both cases, and might be an issue in certain situations.

In the context of [6] for instance, where a central server is used, the full storage technique makes sense, and is more simple and efficient.

Time-Memory Trade-Off. An intermediate solution is the time-memory trade-off (TMTO). The idea is to use memory to reduce the authentication time, making both memory and time suitable to our application. Note that the goal of the TMTO is here to reach an authentication time that is below the acceptable threshold of 200 ms. Once this requirement is fulfilled, still decreasing the time does not make sense because (i) this implies a memory cost (ii) the authentication time would become a negligible factor in the whole communication time.

3 Background on Time-memory Trade-offs

In this section, we briefly recall the required background on the *Time-Memory Trade-off* (TMTO) method. We describe the TMTO technique

[6] If one wants to index the hashes with (i, j) couples, the memory increases by 25% (32 bits appended to each of the 128-bit hashes).

but make no attempt at providing a complete survey of it. For an advanced introduction about this topic we recommend to read [4].

3.1 Introduction

A common search problem in cryptanalysis is finding the preimage of a given output of a one-way function. The first naïve method is applying the function to all possible inputs until finding the expected value. Such an exhaustive search requires N operations in the worst case to find a preimage, where N is the total size of the problem. This becomes impractical when N is large.

The other extreme is to first construct a look-up table including all the preimage values. Afterwards, finding a preimage is done via a table look-up operation which requires a negligible amount of time. The precomputation process requires an effort equal to an exhaustive search, but is to be performed only once. Although this method is quite fast during the online search phase, it may require extreme amounts of memory for large problems.

The comparison of exhaustive search and exhaustive storage methods is depicted in Table 1.

Table 1. Comparison of exhaustive search and table look-up methods (Average case).

	Exhaustive search	Exhaustive storage
Precomputation	0	N
Online computation	$N/2$	0
Memory (storage)	0	N

3.2 Description

The basic idea of the time-memory trade-off (TMTO) method is to find a compromise that has a lower online computation complexity than the exhaustive key search, and a lower memory complexity than the exhaustive storage. Hellman introduced one such trade-off in 1980 [13]. Given a search space of size N, and given M words of memory used for the trade-off, the average number of cryptographic operations T obeys the law $N^2 \propto T \times M^2$ [13]. The principle is the following. During an initial phase, a point is chosen arbitrarily in the search space, hashed (or ciphered, depending on the target function), and then reduced to another

point in the search space. This reduction is the output of a reduction function, which is typically a modulo. This process is iterated a given number of times, forming a chain of hashes. This whole operation is itself repeated many times and only the starting points and endpoints of the chains are kept and stored in a table. Once this table is computed, it is used in the online phase to accelerate the search. The method is probabilistic given that it is very unlikely to fully cover the search space, but several tables can be used to obtain a success probability very close to 1.

A major improvement over Hellman's original TMTO method [13] was given by Oechslin in [8]. The precomputed table called *rainbow table* for this method is structurally different than Hellman's TMTO in that it uses a different reduction function in each column. By doing so, although it might seem to slow the search process, chain fusions (events in the table construction and the search process that degrade the efficiency) in the table are much less frequent and can be detected very easily during their construction. Tables without fusion are said *perfect* [9] and will be used in this paper.

We now give the most relevant results in the analysis of rainbow tables.

Theorem 1. *The probability of success of a set of ℓ rainbow tables of m rows of t columns each, for a problem of size N is:*

$$P = 1 - \left(1 - \frac{m_t}{N}\right)^{\ell t}.$$

Proof. See [4].

Theorem 2. *The maximum number of chains in a rainbow table of t columns, for a problem of size N is:*

$$m_t^{max} = \frac{2N}{t+1}.$$

Proof. See [4].

Theorem 3. *The optimal parameters for a rainbow table, for a problem of size N, given a memory of M and a desired probability of success P^* are:*

$$\begin{aligned} \ell &= \left\lceil \frac{-\log(1-P^*)}{2} \right\rceil, \\ m_t &= \frac{M}{\ell}, \\ t &= \frac{\log(1-P^*)}{\ell \log\left(1 - \frac{m_t}{N}\right)} \approx -\frac{N}{M}\log(1-P^*). \end{aligned}$$

Proof. See [4].

In the following, optimal parameters are implicitly used.

4 OSK/AO

4.1 Description

Avoine and Oechslin propose in [5] to apply the time-memory trade-offs to the search procedure of OSK, leading so to a variant known as OSK/AO. The complexity of the search procedure varies from $O(1)$ to $O(N)$, depending on the amount of memory we are willing to devote to the time-memory trade-off. For example, they mention that a complexity of $O(N^{2/3})$ can be reached with a memory of size $O(N^{2/3})$.

Avoine, Dysli, and Oechslin also suggest in [3] a variant of the OSK protocol that ensures strong authentication, as OSK is originally designed to ensure identification only, without consequently considering replay attacks. To do so, [3] suggests using nonces as follows: the reader sends a nonce r in the authentication request message and the tag answers $\mathcal{G}(S_i^j \oplus r)$ along with $\mathcal{G}(S_i^j)$. The latter value is used by the reader to identify the tag, and the former to authenticate it.

Another advantage of OSK/AO is that the search done in the identification is intrinsically randomized, which makes timing attacks irrelevant [2].

Now we briefly describe the specific time-memory trade-off technique introduced in [3, 5].

In this technique there are two main functions namely a *response generating function* $\mathcal{F}$ and a *reduction function* $\mathcal{R}$. $\mathcal{F}$ takes two indices as an input (i.e., tag index and life time index) and outputs a tag response such that

$$\mathcal{F} : (i, j) \mapsto \mathcal{G}(\mathcal{H}^j(S_i^0)) = r_i^j$$

The reduction function $\mathcal{R}$ is such that

$$\mathcal{R} : r_i^j \mapsto (i', j')$$

where $1 \leq i, i' \leq n$, and $0 \leq j, j' \leq L$.

The main specificity is that $\mathcal{F}$ requires $j+1$ cryptographic operations to be computed, which would drastically lower the efficiency of the search if it were used directly. What is suggested instead is to use a second kind of-time-memory trade-off, called the rapid-hash table, to compute $\mathcal{F}$ efficiently. This trade-off table is rather straightforward: the secrets

S_i^j of the tags are computed from life-time values 0 to L, but only $\frac{L}{\kappa}$ columns are stored. This is illustrated in Fig. 3. As explained in Sect. 4.3, this means that an average of $\frac{\kappa+1}{2}$ cryptographic operations are required per evaluation of $\mathcal{F}$.

$$\underbrace{\begin{array}{ccccccc} S_1^0 & S_0^\kappa & S_0^{2\kappa} & \dots & S_0^{(\lfloor \frac{L}{\kappa} \rfloor - 2)\kappa} & S_0^{(\lfloor \frac{L}{\kappa} \rfloor - 1)\kappa} \\ S_2^0 & S_1^\kappa & S_1^{2\kappa} & \dots & S_1^{(\lfloor \frac{L}{\kappa} \rfloor - 2)\kappa} & S_1^{(\lfloor \frac{L}{\kappa} \rfloor - 1)\kappa} \\ \vdots & & & & & \vdots \\ S_i^0 & S_i^\kappa & S_i^{2\kappa} & \dots & S_i^{(\lfloor \frac{L}{\kappa} \rfloor - 2)\kappa} & S_i^{(\lfloor \frac{L}{\kappa} \rfloor - 1)\kappa} \\ \vdots & & & & & \vdots \\ S_n^0 & S_{n-1}^\kappa & S_{n-1}^{2\kappa} & \dots & S_{n-1}^{(\lfloor \frac{L}{\kappa} \rfloor - 2)\kappa} & S_{n-1}^{(\lfloor \frac{L}{\kappa} \rfloor - 1)\kappa} \end{array}}_{\frac{L}{\kappa} \text{ columns}}$$

Fig. 3. The rapid-hash table.

4.2 Analysis

As explained in Sect. 4.1, there are two things that need to be stored in memory: the rainbow tables and the rapid-hash table. We discuss belowd the proportion of memory that should be dedicated to each.

Let ρ denote the proportion of memory dedicated to the rainbow tables. The trade-off efficiency follows the rule $T = N^2\gamma/M_{RT}^2$ (see [4, 13]), with γ being a small factor depending on the probability of success of the trade-off, and M_{RT} the memory dedicated to the rainbow tables (that is ρM). As for the rapid-hash table, we have:

$$\kappa = \left\lceil \frac{N|hash|}{M_{RH}} \right\rceil,$$

with $|hash|$ the size of a hash, and M_{RH} the memory for the rapid-hash table (that is $(1-\rho)M$). Each operation in the rainbow tables requires an average of $\frac{\kappa+1}{2}$ cryptographic operations in the rapid-hash table. Therefore:

$$T = \frac{N^2\gamma}{M_{RT}^2}\frac{\kappa+1}{2} \approx \frac{N^2\gamma}{\rho^2 M^2}\frac{N|hash|}{2(1-\rho)M}.$$

The optimal value of ρ can be found easily by deriving:

$$\frac{\partial T}{\partial \rho} = 0 \quad \Leftrightarrow \quad \frac{\partial}{\partial \rho}\left[\frac{1}{\rho^2(1-\rho)}\right] = \frac{3\rho - 2}{(\rho-1)^2\rho^3} = 0,$$

which yields $\rho_{opt} = \frac{2}{3}$. In the following, we will thus take the memory for the rainbow tables to be two thirds of the total memory[7].

4.3 Algorithms

We now describe the algorithms used in OSK/AO, namely (i) the algorithm to compute the rapid-hash table (Algorithm 1), (ii) the algorithm to build the TMTO tables (Algorithm 2), and (iii) the algorithm to identify the tag (Algorithm 3). The material in this section mostly comes from [6]. The notations used in the algorithms are given in Table 2.

Table 2. Notations used throughout the paper.

n	Number of tags in the system.
L	Life time of a tag in the system (in terms of authentication executions).
ℓ	Number of TMTO tables.
t	Length of the chains of a rainbow table.
S_i^j	Secret of the i-th tag used for the $j+1$-th authentication where $1 \leq i \leq n$, and $0 \leq j \leq L$.
$\mathcal{H}$, $\mathcal{G}$	Collision resistant one-way functions.
$rapid\mathcal{H}(i,j)$	Function which computes the j-th secret of the i-th tag such that $S_i^j = \mathcal{H}^j(S_i^0)$. This function uses a precomputed rapid hash (RH) table to compute hashes faster. The construction of this function is demonstrated in Algorithm 1.
κ	Length of the interval between hash indices. This parameter is needed for computing rapid hashes.
$state[i][k]$	A pre-computed two-dimensional array which stores the $k \times \kappa$-th hash value of the i-th tag's initial secret ($\mathcal{H}^{k\times\kappa}(S_i^0)$). For instance, let $\kappa = 6$, $i = 1$ and $k = 6$, then $state[1][6]$ stores $S_1^{36} = \mathcal{H}^{36}(S_1^0)$. This array is used during the evaluation of $rapid\mathcal{H}(i,j)$.
$\mathcal{F}(i,j)$	The response generating function inputs two parameters, the tag index and the life time of the tag. This function uses the rapid-hash function. It outputs a tag response such that $\mathcal{F}(i,j) = \mathcal{G}(rapid\mathcal{H}(i,j))$.
$Table_v$	The v-th TMTO table which stores the starting and endpoints (indices) of the TMTO table, where $1 \leq v \leq \ell$.
$\mathcal{R}_w^v(val)$	For w-th column of the v-th table, a simple reduction function which maps input val into a output with smaller size, where $1 \leq w \leq t$.

First, the system randomly generates the initial secrets for all the tags such that $S_i^0 \in_R \{0,1\}^\lambda$ where $1 \leq i \leq n$, and λ is the length of

[7] Note that this result is compliant with the analysis done in [5]. The development done in this section is somewhat simpler and matches the notations used in the rest of this paper.

the secrets. The system defines a κ parameter then computes the interval secret values of all the tags. After that all the secrets are stored into a two dimensional array such that $state[i][k] := \mathcal{H}^{k\times\kappa}(S_i^0)$ where $k = 0, 1, 2, \ldots$ and $0 \leq k \times \kappa \leq L$.

Now, for a given secret of tag i, the j-th rapid-hash computation of the secret is presented in Algorithm 1. The algorithm requires only at most κ hashes by the help of the precomputed RH table. Whenever κ decreases, the memory usage increases but the on-line computation decreases.

Algorithm 1 Compute $y = rapid\mathcal{H}(i, j)$

Require: $1 \leq i \leq n, 0 \leq j \leq L$
Ensure: $y = S_i^j$
  $y \leftarrow state[i][\lfloor \frac{j}{\kappa} \rfloor]$
  $a \leftarrow j \bmod \kappa$
  while $a \neq 0$ **do**
    $y = \mathcal{H}(y)$
    $a \leftarrow a - 1$
  end while
  return y

Algorithm 2 shows the procedure to construct a single rainbow TMTO table. For the construction, only two parameters are needed: the number of starting points used in the precomputation phase (generally named m_1 [4]) and the number of the table to be generated. The starting points of a TMTO table are fed into the $\mathcal{F}$ function sequentially. The output is actually a response of a tag in the system and is fed into the reduction function which outputs arbitrary indices. For a single chain this process is repeated consecutively up to a pre-defined chain size t, then the starting and endpoints are stored in the table. Finally, each generated ending point is compared in the table to detect fusions. When two chains generate a fusion, one of them is discarded. This procedure eventually leads to a perfect table.

Finally, Algorithm 3 shows the identification process of a tag by extracting the pre-image of a given response using TMTO tables. This part of the system runs during the authentication of a tag. First, $TagResp$ (the answer of the tag) is fed into the reduction function R_t^v and searched among the ending points of the TMTO table. (i) If a match is found, the corresponding starting point is iterated as explained in Algorithm 2 up to the $(t-1)^{th}$ reduction function R_t^v in order to get a candidate response. If the candidate response is equal to $TagResp$ then identification

Algorithm 2 Construction of $Table_v$ $(j,\ m_1,\ v)$

Require: $1 \leq j,\ 1 \leq m_1 \leq n \times j\ ,\ v \geq 1$
  $table \leftarrow \{\emptyset\}$
  for $i = 1$ to $\left\lceil \frac{m_1}{j} \right\rceil$ **do**
    for $k = 0$ to j **do**
      $nextResp \leftarrow \mathcal{F}(i, k)$
      for $w = 1$ to $t - 1$ **do**
        $z[\,] \leftarrow \mathcal{R}_w^v(nextResp)$
        $nextResp = \mathcal{F}(z[0], z[1])$
      end for
      $z[\,] \leftarrow \mathcal{R}_t^v(nextResp)$
      if $z \notin table$ **then**
        add the record $\{(i, k); (z[0], z[1])\}$ into $table$
      end if
      if $(i - 1) \times j + k \geq m_1$ **then**
        break
      end if
    end for
  end for
  clean $table$
  return $table$

is completed. Otherwise (ii) $TagResp$ fed into the reduction function such that $R_{t-1}^v(TagResp)$, then the resulting indices fed into $\mathcal{F}$, and then the resulting response fed into $R_t^v(TagResp_{next})$ consecutively. As previously done, the output value search among the endpoints of the TMTO table and the similar process is carried as described above.

5 Experiments and Comparison

5.1 Environment

The precomputations are performed with a personal computer having Intel 2.8GHz Core2 Duo processor, 4GB RAM and Windows 7 - 64-bit operating system. As an NFC enabled mobile phone we use LG OPTIMUS 4X HD having 1.5GHz processor and 1GB RAM [17]. The cell phone has an open source Linux-based operating system, Android. This OS has a large community of contributors who develop applications primarily written in a customized version of the Java programming language [22]. The phone supports both ISO/IEC 14443 and ISO/IEC 15693 standards which are the common standards in order to read/write 13.56 MHz contactless smart cards.

For the tags, we work on professional version of ZeitControlers basic card $ZC7.5$ $(ZC - Basic)$ which is a programmable processor card as

Algorithm 3 Identify ($Table_v$, TagResp)

Require: TagResp $\in \{0,1\}^\lambda$, $v \geq 1$
Ensure: TagResp $\leftarrow \mathcal{G}(y)$
  for $q = t$ down to 1 **do**
    $nextResp \leftarrow$TagResp
    for $i = q$ to $t-1$ **do**
      $z[\,] \leftarrow \mathcal{R}_i^v(nextResp)$
      $nextResp \leftarrow \mathcal{F}(z[0], z[1])$
    end for
    $z[\,] \leftarrow \mathcal{R}_t^v(nextResp)$
    if $z \in Table_v$ **then**
      $\{z'; z\} \leftarrow Table_v(z)$
      $nextResp \leftarrow \mathcal{F}(z'[0], z'[1])$
      for $w = 1$ to $q-1$ **do**
        $\widetilde{z}[\,] \leftarrow \mathcal{R}_w^v(nextResp)$
        $nextResp \leftarrow \mathcal{F}(\widetilde{z}[0], \widetilde{z}[1])$
      end for
      if $nextResp = TagResp$ **then**
        return $true$
      end if
    end if
  end for
  return $false$

hardware environment for protocol implementation [12]. It has a microcontroller with 32kB user EEPROM that holds its own operating system (OS) and it has 2.9kB RAM for user tag's data. It supports ISO/IEC 14443. The EEPROM contains the user's Basic code, compiled into a virtual machine language known as P-Code (the Java programming language uses the same technology). The RAM contains run-time data and the P-Code stack. The overview of the system is depicted in Figure 4.

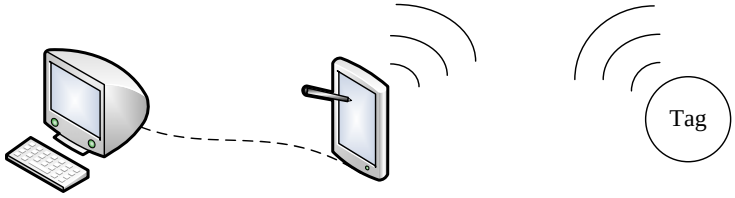
5.2 Parameters and Functions

The parameters for the experiments are $n = 2^{20}$, $L = 2^7$ and the one-way functions we selected are the following ones[8]:

- $\mathcal{H}(S_i^j) : AES_K(S_i^j) \oplus S_i^j = S_i^{j+1}$,
- $\mathcal{G}(S_i^j) : AES_K(S_i^j + 1) \oplus (S_i^j + 1) = r_i^j$

where K is a 128-bit constant key. This is known as the Matyas-Meyer-Oseas construction [19]. Its goal is to build a one-way function from a block cipher.

[8] The parameters are the same than the ones in [3].



Processor: 2.8 GHz
RAM: 4 GB
Windows 7 – 64 bit
Programming Lang: Java

LG Optimus 4X P880
Android 4.1
NFC enabled phone
Processor: 1.5 GHz
RAM: 1.0 GB

Basic card ZC 7.5
EEPROM: 32 kB
RAM: 2.9 kB

Fig. 4. Overview of the system

We use the AES algorithm in the construction because it is commonly implemented on fewer gates than classical hash functions (see e.g. [11]), and, in particular, is also available in the ZC7.5. This construction requires only one key schedule during the initialization phase of the tags, which makes algorithm faster.

To construct rainbow tables each column of each table uses a different reduction function. The function takes three parameters that are the table index ($v \in [0, 1, \ldots, \ell - 1]$), the column index ($w \in [1, 2, \ldots, t]$) and the response output as a byte array ($val[.]$). This function produces two output values; the first one is for tag index ($i = 0, \ldots, n - 1$), the second one is for lifetime index($j = 0, \ldots, L - 1$). The i value is computed as $i = (Int32(val[v, v+3]) + w) \mod n$ where the function $Int32$ converts a given input 4-byte array into an unsigned 32-bit integer. The j value is computed as $i = (Int32(val[v+1, v+4]) + w) \mod L$. The construction of our reduction functions are given in Algorithm 4.

Algorithm 4 Compute $\mathcal{R}_w^v(val[.])$

Require: $v \geq 0$, $w \geq 1$
Ensure: $i \in \mathbb{Z}_n$, $j \in \mathbb{Z}_L$
  $i \leftarrow Int32(val[v, v + 3]) + w$
  $j \leftarrow Int32(val[v + 1, v + 4]) + w$
  $i = i \ mod \ n$
  $j = j \ mod \ L$
  return $\{i, j\}$

5.3 Precomputation of the Tables

In order to use our implementation on low-resource devices (such as handheld readers, PDAs and NFC compliant cellphones) we build tables that can fit to small RAMs.

For the total memory there are two parts: (i) the rapid hash table that stores some intermediate values of the OSK table and (ii) the TMTO tables[9]. We use the optimal parameters, so we compute the κ and t such that the memory consumption is as described in 4.2.

Another significant choice for the TMTO construction is the probability of success. It should be high enough to avoid false negatives during the authentication process. In our scenario, using $\ell = 4$ rainbow tables of maximal size, the probability to identify a tag is greater than 0.999 according to Theorem 1. Note that trying to reach a higher success probability does not make sense given that the probability of failure due to noise on the channel is even higher.

Finally, regarding the number of starting points m_1, we use the same trick as in [4] to reduce the precomputation effort. In our case, we obtain about 98% of the maximal number of ending points by starting with 50 times that number.

In total, the precomputation cost is $\ell \times m_1 \times t$ evaluations of $\mathcal{F}$, which is about $4 \times 50 \times m_t \times t = 400nL$ in our case (see Theorem 2). Since these are $\mathcal{F}$ evaluations, this number is also multiplied by $\frac{\kappa+1}{2}$ hash operations. For instance, if $\kappa = 6$ and on a server capable of 2^{20} hash operations per second, the precomputation stage would take about 50 hours. Some details about the precomputation of rainbow tables seem to have been overlooked in [3, 5], which would explain their optimistic result. However, we can do much better than that if we build a table containing the nL secrets, and use it during the precomputation instead of the *rapidH* table. This table needs $nL|hash|$ bits, that is 2GB in our case, and takes about 2 minutes to build on the server. In this case, there are actually no hash operations during the building of the TMTO table, making this procedure faster. In our case the whole precomputation process takes about an hour.

5.4 Experiments

We tested the performance in two settings by running Algorithm 3 (i.e., identification process of OSK/AO with randomly chosen tags). Our mobile phone [17] is able to compute about 187,750 hashes per second. For

[9] We used the prefix-suffix decomposition method, as described for instance in [7] in order to reduce to some extent the size of the TMTO tables.

both settings, the experiment is run $1,000,000$ times. The experimental results are depicted in Table 3.

Table 3. Results of experiments on an NFC compliant cellphone

Memory	253MB	113MB
Identification time	15.26ms	117.54ms
Length of the chains of the TMTO (t)	27	72
Number of chains of the TMTO (m_t)	8968214	3566605
Rapid-hash parameter(κ)	22	43
Authentication rate	99.9%	99.9%

We also measure the time when we use our system with a real tag. There are three phases on the tag's side: receiving a query, computing the response (two hash calculations), and sending the response. The total time is 70 ms on average, including 50 ms for the calculation of the two hash values and 20 ms for the communication.

It can be seen that the average identification time is below the 200 ms threshold (if we include the 70 ms for the tag computation and the communication) even for a memory below 128MB. We thus show that one can achieve very fast authentication even with limited memory.

6 Conclusion

We have implemented the OSK/AO [3] protocol on an NFC-compliant cellphone and a ZC7.5 contactless tag. Our implementation is fully operational and is, to the best of our knowledge, the first implementation of a privacy-friendly authentication protocol based on symmetric-key cryptography. The implementation is suited to large-scale applications, e.g. a million of tags, as this can be the case in mass transportation systems, even on low-resource mobile devices such as hand held readers, PDAs or NFC compliant cellphones. We have run several experiments on the implemented RFID system and we show that the results obtained match the theory and are favorable to a practical deployment.

Acknowledgements

This work is partially funded by the Walloon Region Marshall plan through the SPW DG06 Project TRASILUX.

References

1. Gildas Avoine, Muhammed Ali Bingöl, Xavier Carpent, and Sıddıka Berna Ors Yalcin. Privacy-friendly authentication in RFID systems: On sub-linear protocols based on symmetric-key cryptography. *IEEE Transactions on Mobile Computing*, preprint, September 2012.
2. Gildas Avoine, Iwen Coisel, and Tania Martin. Time Measurement Threatens Privacy-Friendly RFID Authentication Protocols. In S.B. Ors Yalcin, editor, *Workshop on RFID Security – RFIDSec'10*, volume 6370 of *Lecture Notes in Computer Science*, pages 138–157, Istanbul, Turkey, June 2010. Springer.
3. Gildas Avoine, Etienne Dysli, and Philippe Oechslin. Reducing Time Complexity in RFID Systems. In Bart Preneel and Stafford Tavares, editors, *Selected Areas in Cryptography – SAC 2005*, volume 3897 of *Lecture Notes in Computer Science*, pages 291–306, Kingston, Canada, August 2005. Springer.
4. Gildas Avoine, Pascal Junod, and Philippe Oechslin. Characterization and Improvement of Time-Memory Trade-Off Based on Perfect Tables. *ACM Trans. Inf. Syst. Secur.*, 11:17:1–17:22, July 2008.
5. Gildas Avoine and Philippe Oechslin. A Scalable and Provably Secure Hash Based RFID Protocol. In *International Workshop on Pervasive Computing and Communication Security – PerSec 2005*, pages 110–114, Kauai Island, HI, USA, March 2005. IEEE, IEEE Computer Society.
6. Muhammed Ali Bingöl. Security analysis of RFID authentication protocols based on symmetric cryptography and implementation of a forward private scheme. Master's thesis, Istanbul Technical University, Istanbul, Turkey, January 2012.
7. Alex Biryukov, Adi Shamir, and David Wagner. Real time cryptanalysis of A5/1 on a PC. In Bruce Schneier, editor, *Fast Software Encryption – FSE'00*, volume 1978 of *Lecture Notes in Computer Science*, pages 1–18, New York, USA, April 2000. Springer-Verlag.
8. Dan Boneh, editor. *Advances in Cryptology - CRYPTO 2003, 23rd Annual International Cryptology Conference*, volume 2729 of *Lecture Notes in Computer Science*, Santa Barbara, California, USA, August 2003. Springer.
9. Johan Borst, Bart Preneel, and Joos Vandewalle. On the Time-Memory Tradeoff Between Exhaustive Key Search and Table Precomputation. In *Proceeding of the 19th Symposium in Information Theory in the Benelux, WIC*, pages 111–118, Veldhoven, The Netherlands, 1998.
10. HID Global Corporation. HSPD-12 & FIPS 201 PIV II: How Government Standards Affect Physical Access Control. http://www.hidglobal.com/sites/hidglobal.com/files/hid-how-gov-stanards-affect-physical-access-control-wp-en.pdf, 2007.
11. Martin Feldhofer, Johannes Wolkerstorfer, and Vincent Rijmen. AES Implementation on a Grain of Sand. *IEE Proceedings – Information Security*, 152(1):13–20, October 2005.
12. Tony Guilfoyle. The zeitcontrol basiccard family. http://www.basiccard.com, 2009.
13. Martin Hellman. A cryptanalytic time-memory trade-off. *IEEE Transactions on Information Theory*, 26(4):401–406, 1980.
14. International Organization for Standardization. ISO/IEC 9798: Information technology – Security techniques – Entity authentication – Part 2: Mechanisms using symmetric encipherment algorithms, 1999.
15. Ari Juels and Stephen Weis. Defining Strong Privacy for RFID. In *International Conference on Pervasive Computing and Communications – PerCom 2007*, pages 342–347, New York City, NY, USA, March 2007. IEEE, IEEE Computer Society.

16. Süleyman Kardaş, Albert Levi, and Ertugrul Murat. Providing Resistance against Server Information Leakage in RFID Systems. In *New Technologies, Mobility and Security – NTMS'11*, pages 1–7, Paris, France, February 2011. IEEE, IEEE Computer Society.
17. LG OPTIMUS 4X HD P880. Technical Specifications. http://www.lg.com/uk/mobile-phones/lg-P880/technical-specifications, 2013.
18. Chae Hoon Lim and Taekyoung Kwon. Strong and Robust RFID Authentication Enabling Perfect Ownership Transfer. In Peng Ning, Sihan Qing, and Ninghui Li, editors, *International Conference on Information and Communications Security – ICICS'06*, volume 4307 of *Lecture Notes in Computer Science*, pages 1–20, Raleigh, North Carolina, USA, December 2006. Springer.
19. Stephen M. Matyas, Carl H. Meyer, and Jonathan Oseas. Generating strong one-way functions with cryptographic algorithm. *IBM Technical Disclosure Bulletin*, 27(10A):5658–5659, 1985.
20. Miyako Ohkubo, Koutarou Suzuki, and Shingo Kinoshita. Cryptographic Approach to "Privacy-Friendly" Tags. In *RFID Privacy Workshop*, MIT, MA, USA, November 2003.
21. Raphael C.-W. Phan, Jiang Wu, Khaled Ouafi, and Douglas R. Stinson. Privacy analysis of forward and backward untraceable rfid authentication schemes. *Wirel. Pers. Commun.*, 61(1):69–81, November 2011.
22. Shankland, Stephen. Google's Android parts ways with Java industry group. CNET News. Retrieved 2012-02-15, November 12, 2007.